allinea

Leaders in parallel software development tools

Session 1: Use The Source

allinea
MAP

www.allinea.com

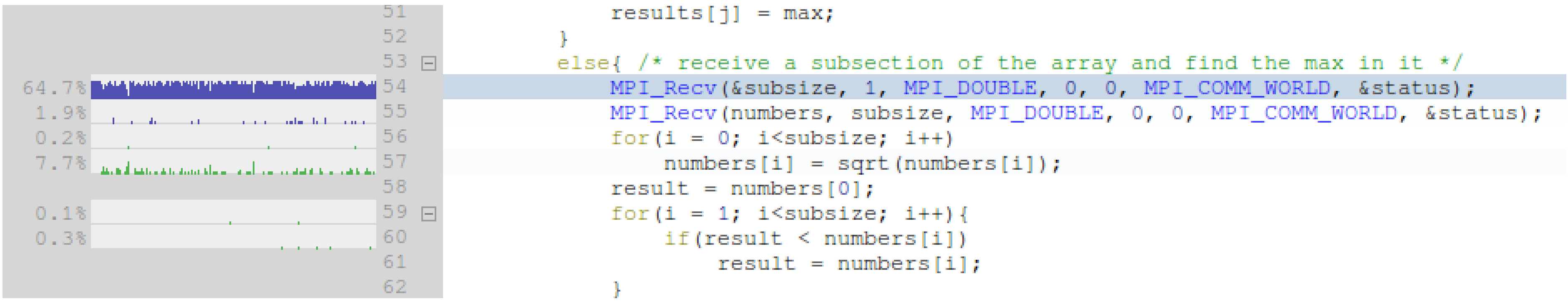
Session 1: Use The Source

-> Measure the performance of an MPI program

-> Explore the code with Allinea MAP's viewer

-> Identify a bottleneck and deduce its cause

-> Change the code and measure the improvement



```
            results[j] = max;
        }
        else{ /* receive a subsection of the array and find the max in it */
            MPI_Recv(&subsize, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(numbers, subsize, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            for(i = 0; i<subsize; i++)
                numbers[i] = sqrt(numbers[i]);
            result = numbers[0];
            for(i = 1; i<subsize; i++){
                if(result < numbers[i])
                    result = numbers[i];
            }
```

Review: Use The Source

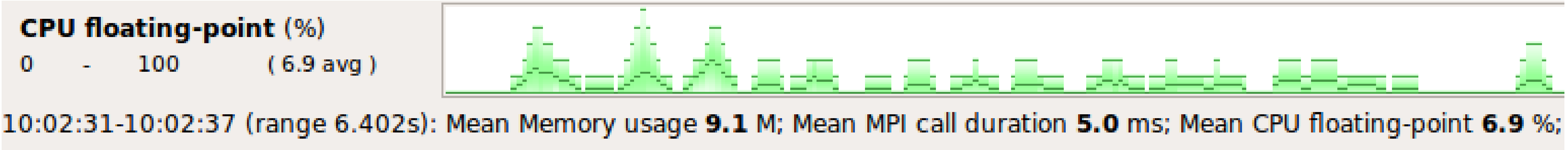
-> Introduced to Allinea MAP

-> Used code folding to explore an unfamiliar file

-> Saw high MPI usage and high MPI imbalance

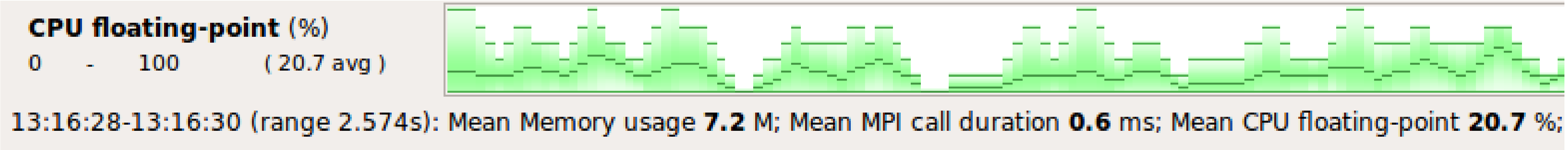
-> Deduced cause – rand() is slow!

-> Fixed by distributing rand() work to nodes for >2x speedup



CPU floating-point (%)
0 - 100 (6.9 avg)

10:02:31-10:02:37 (range 6.402s): Mean Memory usage **9.1** M; Mean MPI call duration **5.0** ms; Mean CPU floating-point **6.9** %;



CPU floating-point (%)
0 - 100 (20.7 avg)

13:16:28-13:16:30 (range 2.574s): Mean Memory usage **7.2** M; Mean MPI call duration **0.6** ms; Mean CPU floating-point **20.7** %;

allinea

Leaders in parallel software development tools

Session 2: CPU Optimization

allinea
MAP

Session 2: CPU Optimization

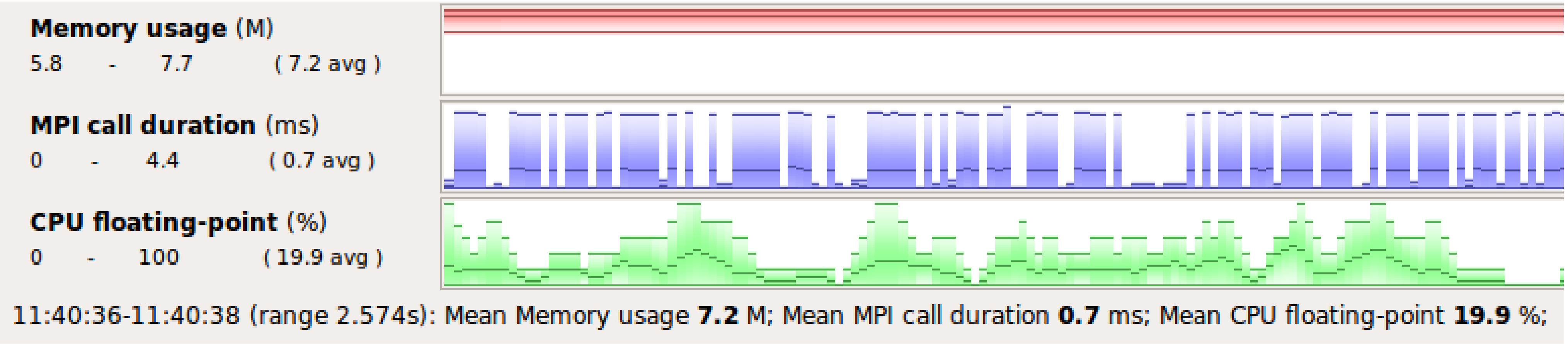
-> Interpret Allinea MAP's metric graphs

-> Explore compiler loop vectorization

-> Recognize cache-related problems

-> Improve loop cache performance

-> Experiment with conflicting optimizations



Memory usage (M)
5.8 - 7.7 (7.2 avg)

MPI call duration (ms)
0 - 4.4 (0.7 avg)

CPU floating-point (%)
0 - 100 (19.9 avg)

11:40:36-11:40:38 (range 2.574s): Mean Memory usage **7.2** M; Mean MPI call duration **0.7** ms; Mean CPU floating-point **19.9** %;

Review: CPU Optimization

-> Introduced to single-core optimization with Allinea MAP

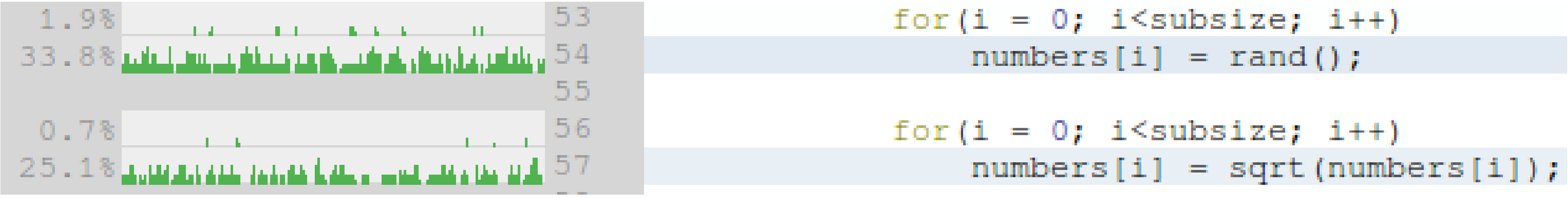
-> Saw zero vectorization and corrected with compiler flags

-> Recognized poor cache performance and its solutions:

In this case improved temporal locality with loop fusion

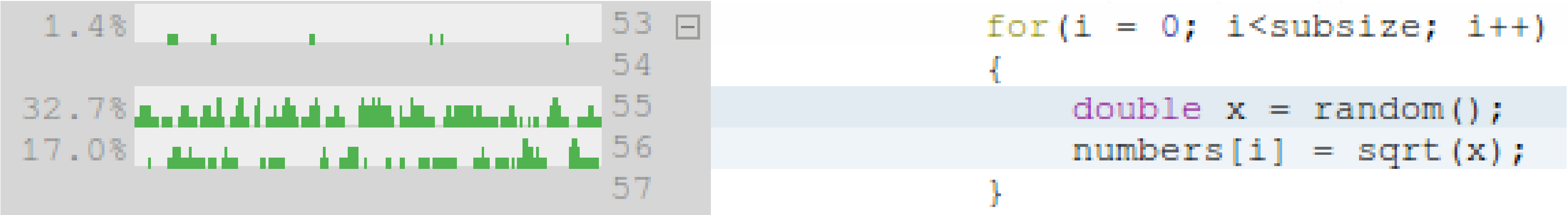
-> Looked at conflicting optimizations – vectorization and loop fusion

-> Found further benefits by swapping library functions



```
for(i = 0; i<subsize; i++)
    numbers[i] = rand();

for(i = 0; i<subsize; i++)
    numbers[i] = sqrt(numbers[i]);
```



```
for(i = 0; i<subsize; i++)
{
    double x = random();
    numbers[i] = sqrt(x);
}
```

Review: Profiling with Allinea MAP

Compile with both -O3 and -g

    -ffast-math and friends are also recommended!

Remember that “time mpirun” includes system overheads

Run interactively with: map *program-name*

Run in batch mode with: map -n *#procs* -profile *program-name*

Use View->Fold All to explore unfamiliar files

Use metric views to spot imbalance and cause of bottlenecks

In this course we improved performance by **4x** – let me know how you get on with your own codes!